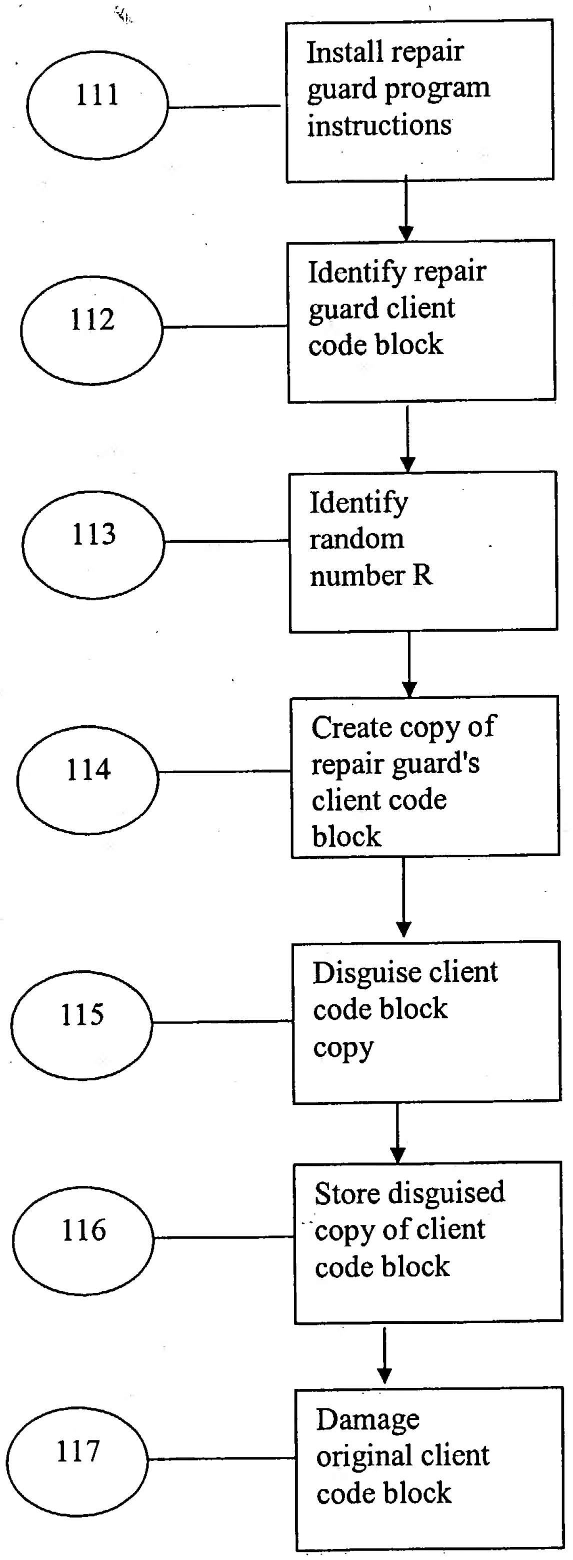

111
Install repair guard program instructions
112
Identify repair guard client code block
113
Identify random number R
114
Create copy of repair guard's client code block
115
Disguise client code block copy
116
Store disguised copy of client code block
117
Damage original client code block

FIG. 1A

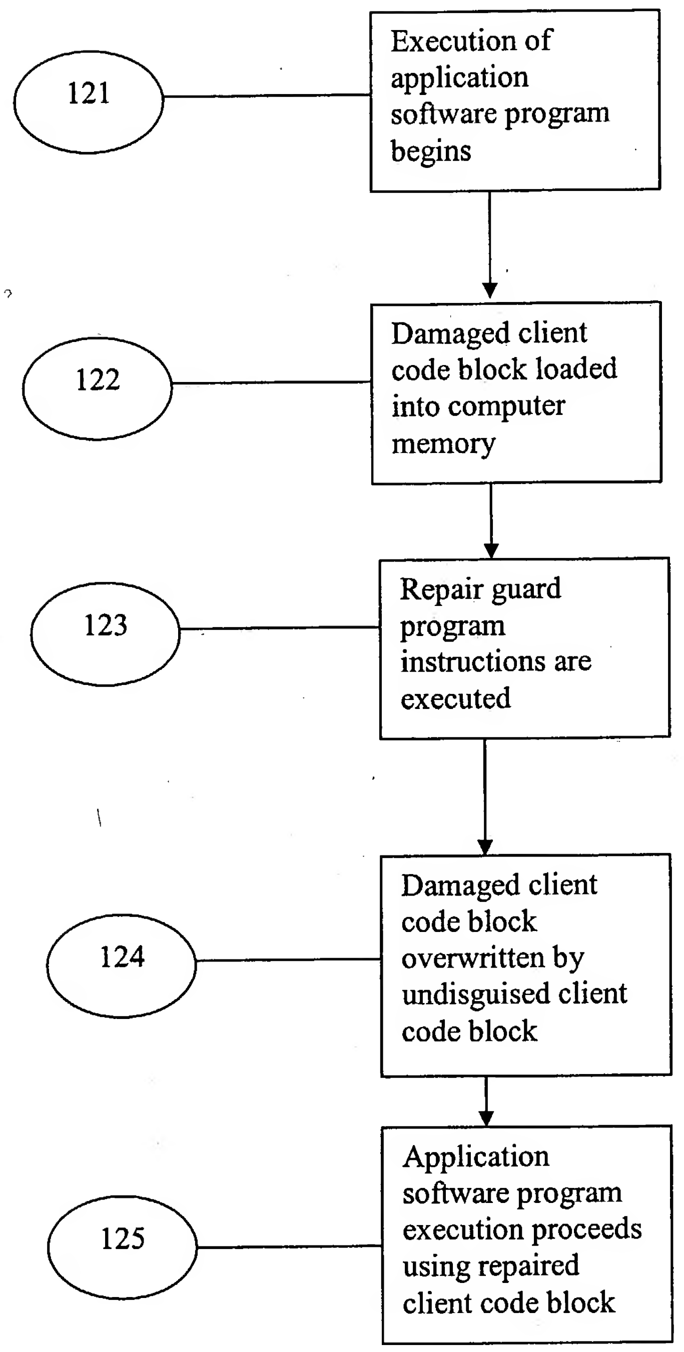
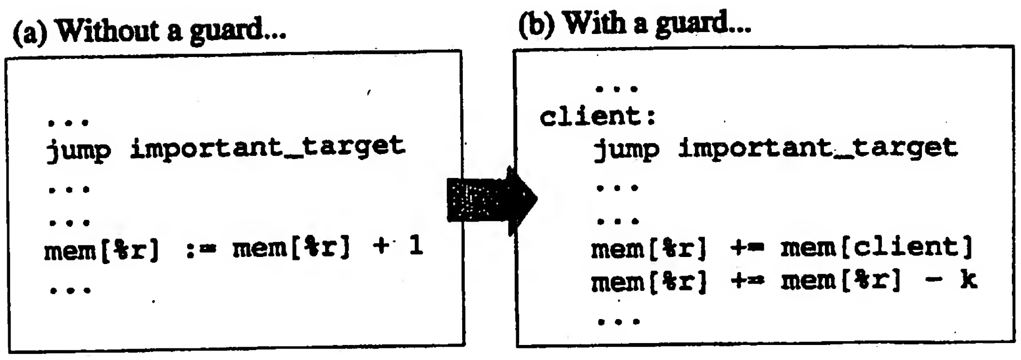

121
Execution of application software program begins
122
Damaged client code block loaded into computer memory
123
Repair guard program instructions are executed
124
Damaged client code block overwritten by undisguised client code block
125
Application software program execution proceeds using repaired client code block

FIG. 1B



```
...
jump important_target
...
...
mem[%r] := mem[%r] + 1
...
```

```
   ...
client:
   jump important_target
   ...
   ...
   mem[%r] += mem[client]
   mem[%r] += mem[%r] - k
   ...
```

FIG. 2A

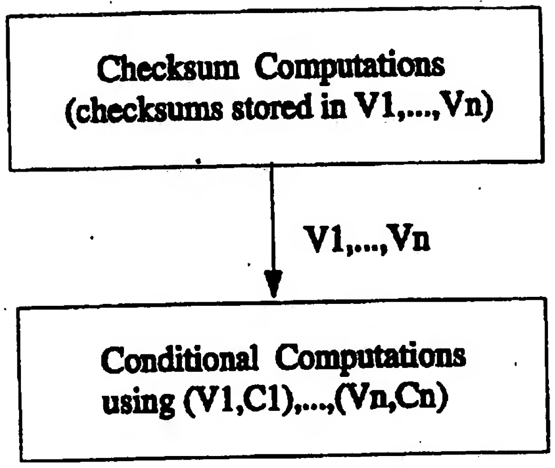

Checksum Computations
(checksums stored in V1,...,Vn)
V1,...,Vn
Conditional Computations
using (V1,C1),...,(Vn,Cn)

FIG. 2B

```
// template 1

        movl        $RANDOM_1, CHECKSUM_1 // RANDOM_1 = any integer
        movl        $START_1, TEMP_1
LABEL_1:
        Cmpl        $END_1, TEMP_1
        jg          LABEL_2
        addl        (TEMP_1), CHECKSUM_1
        addl        $RANDOM_2, TEMP_1 // RANDOM_2 in [3,5]
        jmp         LABEL_1
LABEL_2:

//template2

        movl        $START_1+END_1+RANDOM_1, TEMP_1 // RANDOM_1 = any
                    integer
        xorl        CHECKSUM_1, CHECKSUM_1
        movl        TEMP_1, CHECKSUM_2
LABEL_1:
        addl        -END_1-RANDOM_1(TEMP_1), CHECKSUM_1
        xorl        -END_1-RANDOM_2+3(TEMP_1), CHECKSUM_2
        subl        $-RANDOM_2, TEMP_1// RANDOM_2 in [3,5]
        cmpl        $END_1+END_1+RANDOM_1, TEMP_1
        jle         LABEL_1
```

FIG. 3

$$
\begin{array}{lll}
(x + 3)(y + 7) & & (1) \\
= \quad (x + 3 + 0)(y + 7 + 0) & & (2) \\
= \quad (x + 3 + u - u_0)(y + 7 + w_0 - w) & \text{iff } u = u_0 \text{ and } w = w_0 & (3) \\
= \quad (x + u + k_1)(y - w + k_2) & \text{where } k_1 = 3 - u0 \text{ and } k_2 = 7 + w_0 & (4)
\end{array}
$$

FIG. 4

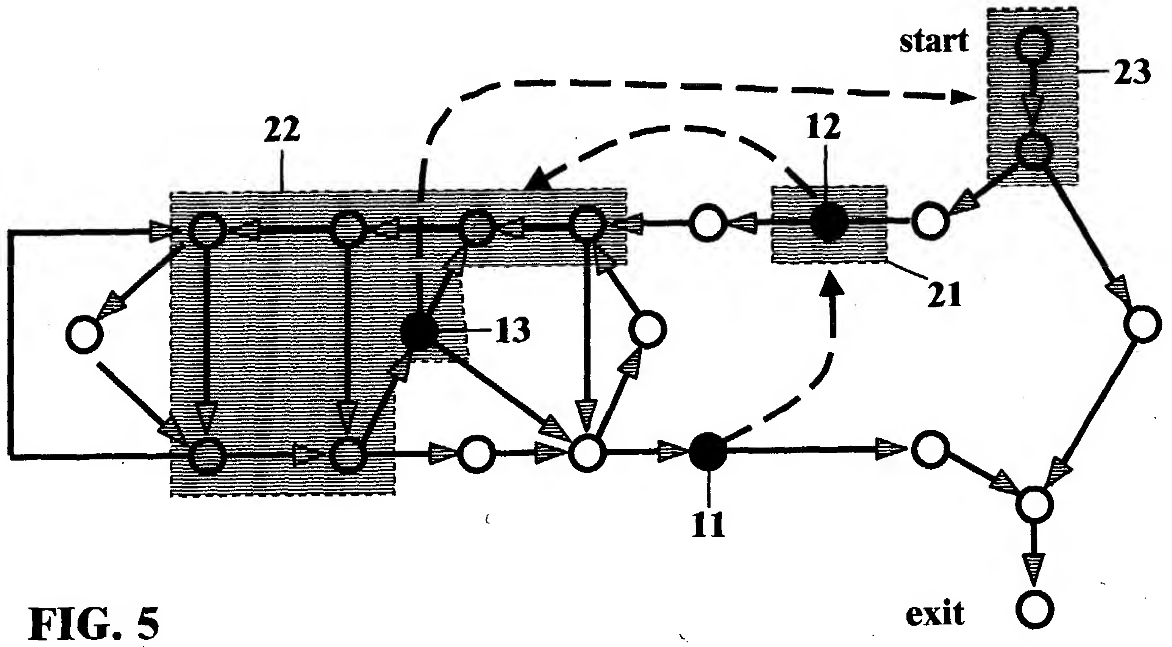


FIG. 5

```
main:
            leal      -4(%esp), %esp         // %esp := %esp – 4
            movl      %ebp, (%esp)           // mem[%esp] := %ebp
            movl      %esp, %ebp
            subl      $8, %esp               // %esp := %esp – 8
            leal      -8(%ebp), %eax
            leal      -4(%esp), %esp
            movl      %eax, (%esp)
            leal      -4(%esp), %esp
            movl      $str1, (%esp)
            leal      -4(%esp), %esp
            movl      $next1, (%esp)         // mem[%esp] := $next1
            jmp       scanf                  //jump to location scanf

next1:
            addl      $8, %esp               // %esp :=%esp + 8
            leal      -4(%esp), %esp
            movl      -8(%ebp), %eax         // %eax := mem[%ebp-8]
            movl      %eax, (%esp)
            leal      -4(%esp), %esp
            movl      $next2, (%esp)
            jmp       pr_fact

next2
            addl      $4, %esp
            movl      %ebp, %esp
            movl      (%esp), %ebp
            leal      4(%esp), %esp
            leal      4(%esp), %esp
            jmp       *-4(%esp)

pr_fact:
            leal      -4(%esp), %esp
            movl      %ebp, (%esp)
            movl      %esp, %ebp
            subl      $4, %esp
            movl      $1, -4(%ebp)
            jmp       .L94

.L94
            cmpl      $1, 8(%ebp)            // mem[%ebp+8] – 1 == ??
            g         .L96                   // if ?? > 0, jmp .L96
            jmp       .L98

.L96
            movl      -4(%ebp), %eax
            imul      8(%ebp), %eax          // %eax := %eax * mem[%ebp+8]
            movl      %eax, -4(%ebp)
            subl      $1, 8(%ebp)
            jmp       .L94

.L98
            leal      -4(%esp), %esp
            movl      -4(%ebp), %eax
            movl      %eax, (%esp)
            leal      -4(%esp),
```

FIG. 6A

```
            movl    $str2, (%esp)
            leal    -4(%esp), %esp
            movl    $next3, (%esp)
            jmp     printf

next3:
            addl    $8, %esp
            movl    %ebp, %esp
            movl    (%esp), %ebp
            leal    4(%esp), %esp
            leal    4(%esp), %esp
            jmp     *-4(%esp)                   // jump to addr in mem[%esp-4]

```

FIG. 6B

```
main:
            leal        -4(%esp), %esp
            movl        %ebp, (%esp)
            movl        %esp, %ebp
            subl        $8, %esp
            leal        -8(%ebp), %eax
            leal        -4(%esp), %esp
            movl        %eax, (%esp)
            leal        -4(%esp), %esp
            movl        $str1, (%esp)
            leal        -4(%esp), %esp
            movl        $next1, (%esp)
            jmp         scanf

next1:                                          // start of client
            addl        $8, %esp
            leal        -4(%esp), %esp
            movl        -8(%ebp), %eax
            movl        %eax, (%esp)
            leal        -4(%esp), %esp
            movl        $next2, (%esp)
            jmp         pr_fact
end1:                                           // end of client

next2:
            addl        $4, %esp
            movl        %ebp, %esp
            movl        (%esp), %ebp
            leal        4(%esxp), %esp
            leal        4(%esp), %esp
            jmp         *-4(%esp)

pr_fact:
            leal        -4(%esp), %esp
            movl        %ebp, (%esp)
            movl        %esp, %ebp
            subl        $4, %esp

            // guard installation site

            movl        $100, g1
            movl        $next1, %eax
guard1_1:
            cmpl        $end1, %eax
            jg          guard1_2
            movl        g1, %ecx
            addl        (%eax), %ecx
            movl        %ecx, g1
            addl        $3, %eax
            jmp         guard1_1
guard1_2:

            // end of checksumming: (g1, G!)

            movl        $-G1+1, %eax            // G1 is the checksum constant
            addl        g1, %eax
```

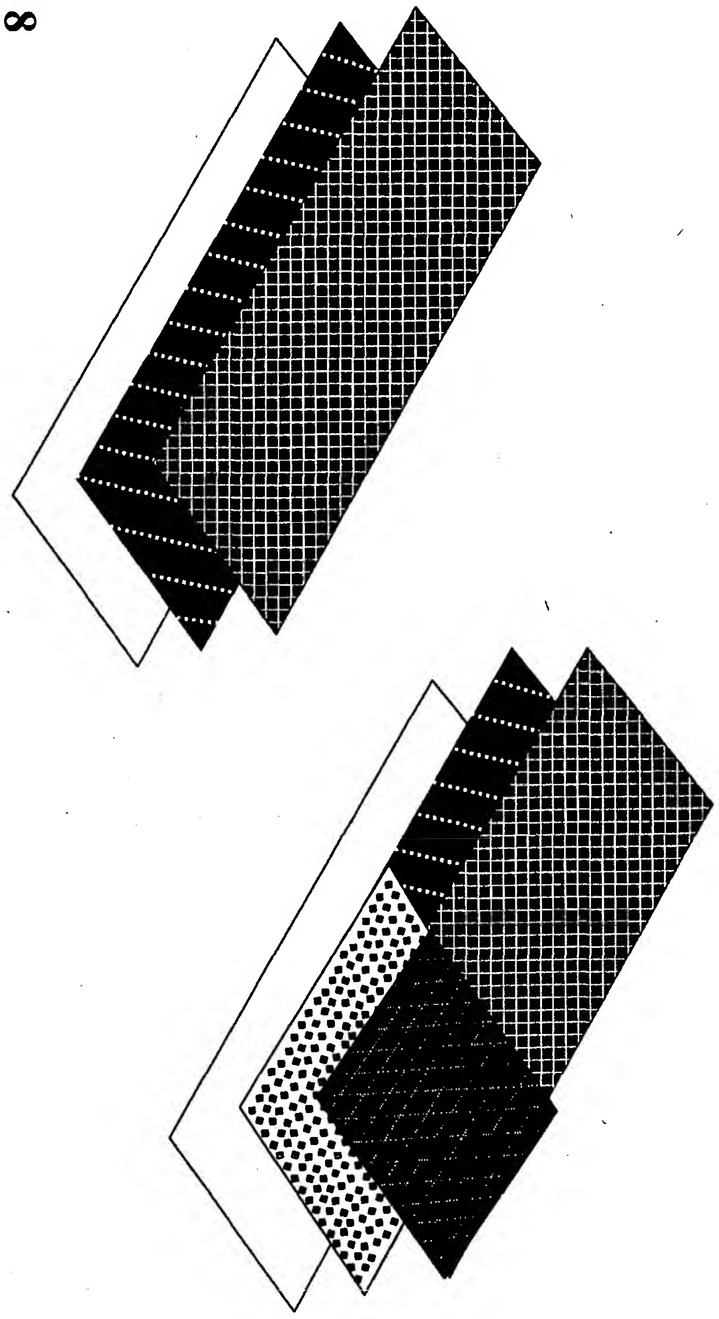
FIG. 7A

```
                movl    %eax, -4(%ebp)
                jmp     .L94

.L94:
                leal    8+G1(%ebp), %eax
                subl    g1, %eax
                movl    (%eax), %eax
                cmpl    $1, %eax
                jg      .L96
                jmp     .L98

.L96:
                movl    -4(%ebp), %eax
                imul    8(%ebp), %eax
                movl    %eax, -4(%ebp)
                subl    $1, 8(%ebp)
                jmp     .L94

.L98:
                leal    -4(%esp), %esp
                movl    -4(%ebp), %eax
                movl    %eax, (%esp)
                leal    -4(%esp), %esp
                movl    $str2, (%esp)
                leal    -4(%esp), %esp
                movl    $next3, (%esp)
                jmp     printf

next3:
                addl    $8, %esp
                movl    %ebp, %esp
                movl    (%esp), %ebp
                leal    4(%esp), %esp
                leal    4(%esp), %esp
                jmp     *-4(%esp)

```

FIG. 7B

EJ915569813US
P00620US00 (19232.0003)

FIG. 8



EJ915569813US
P00620US00 (19232.0003)

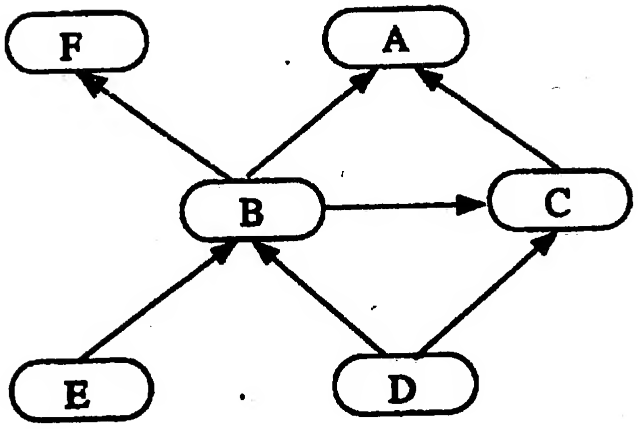


FIG. 9 A

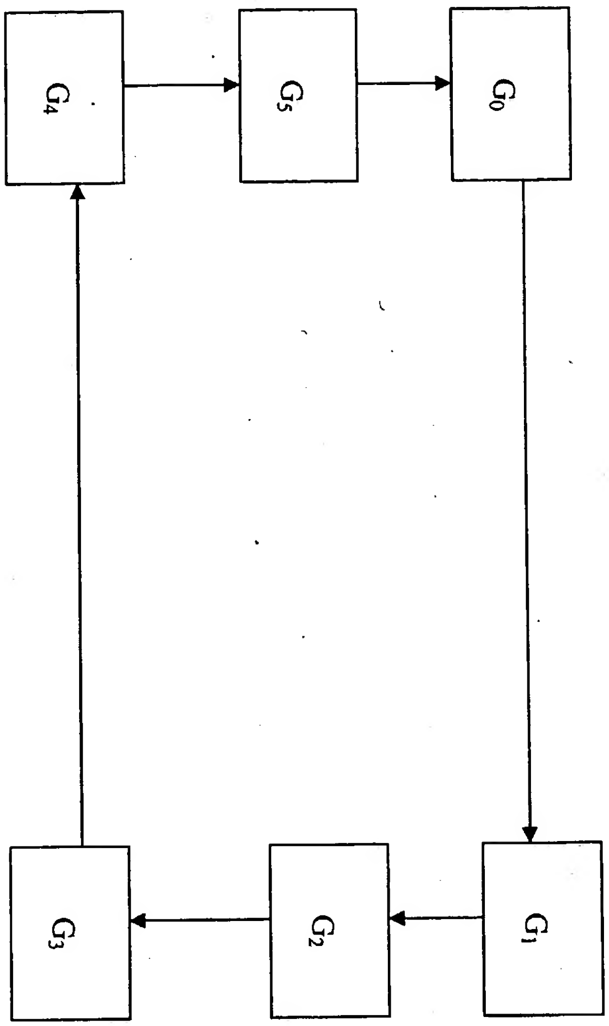


FIG. 9B

EJ915569813US
P00620US00 (19232.0003)

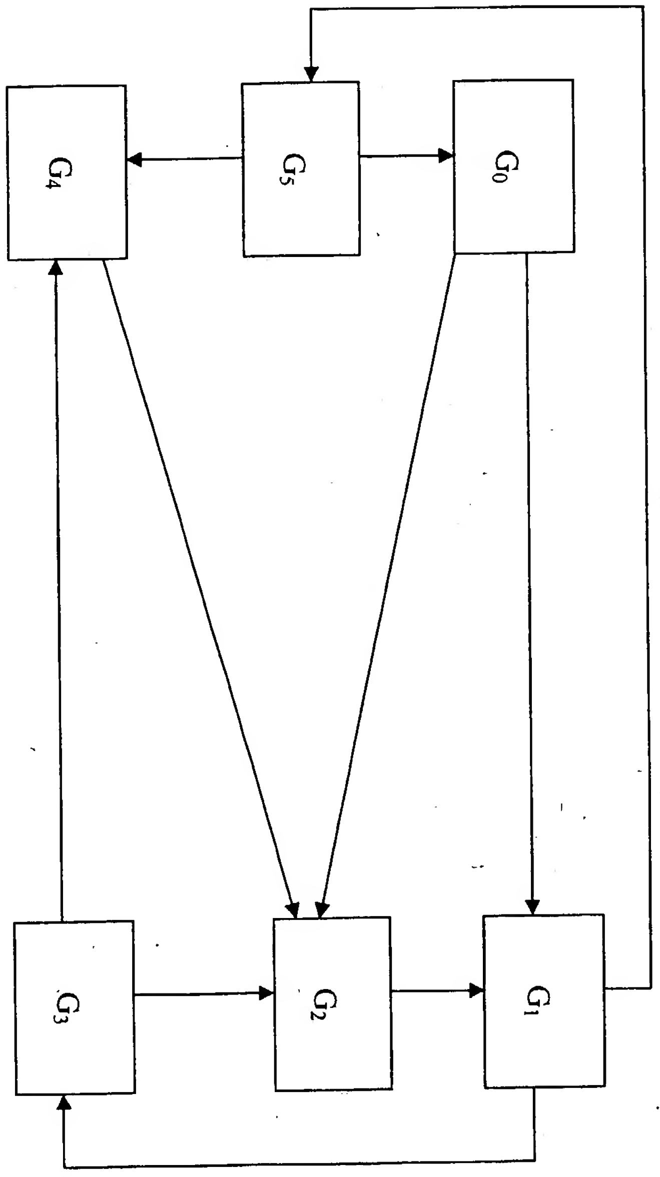


FIG. 9C

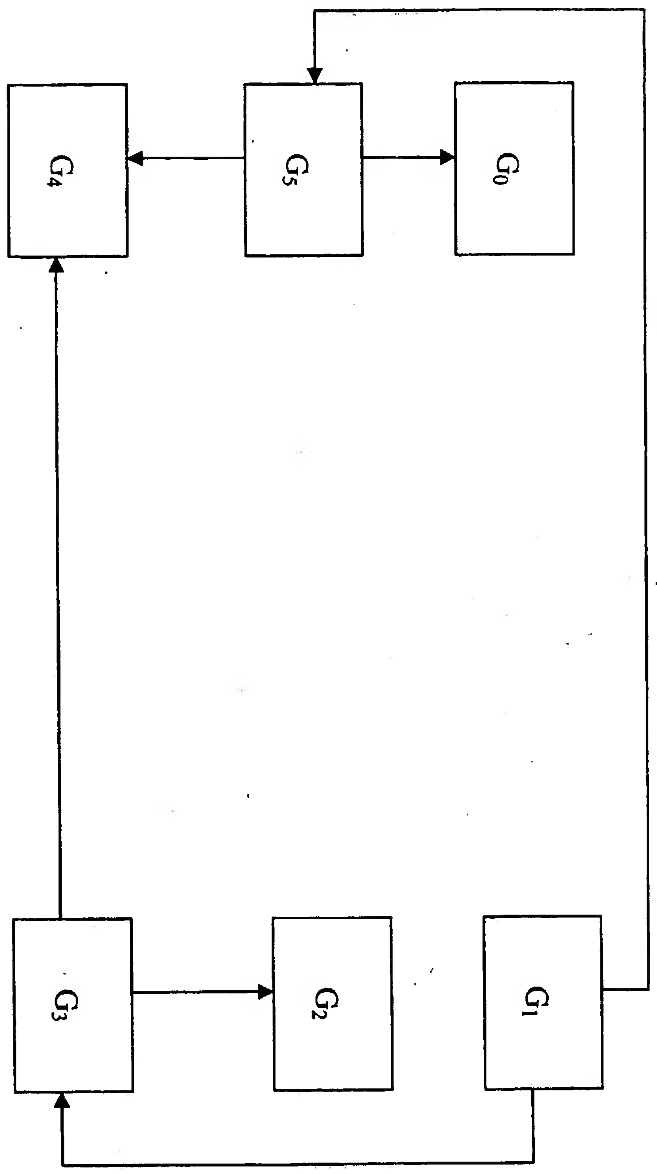


FIG. 9D

EJ915569813US
P00620US00 (19232.0003)

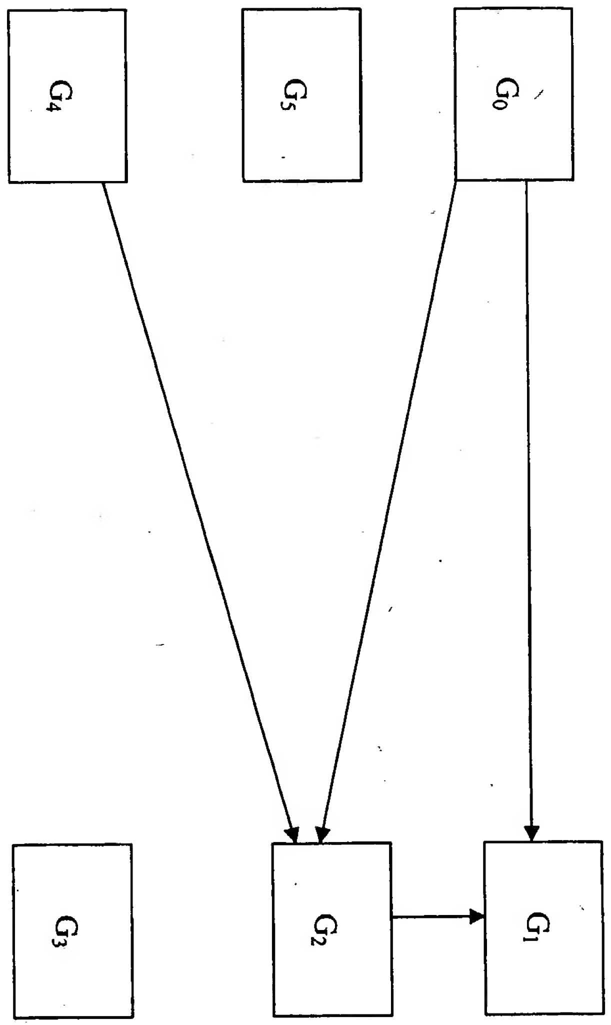


FIG. 9E

EJ915569813US
P00620US00 (19232.0003)

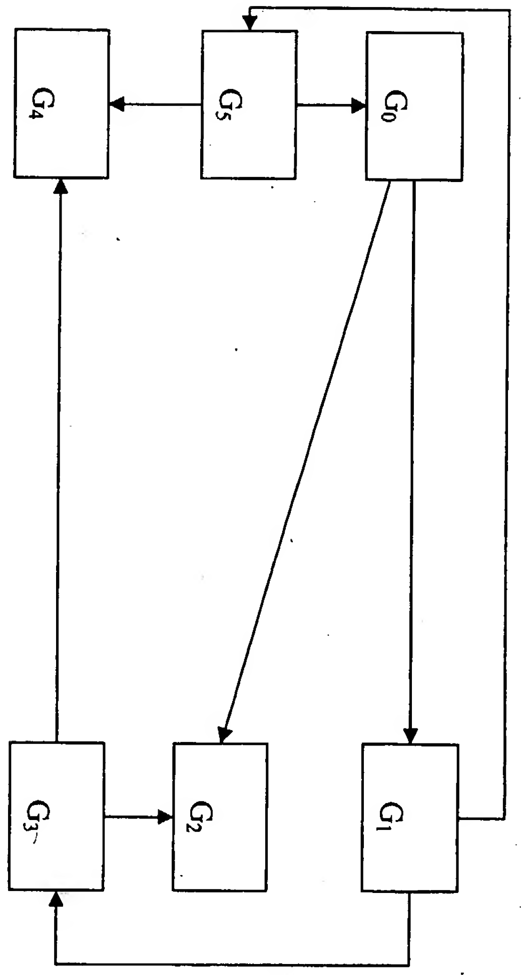
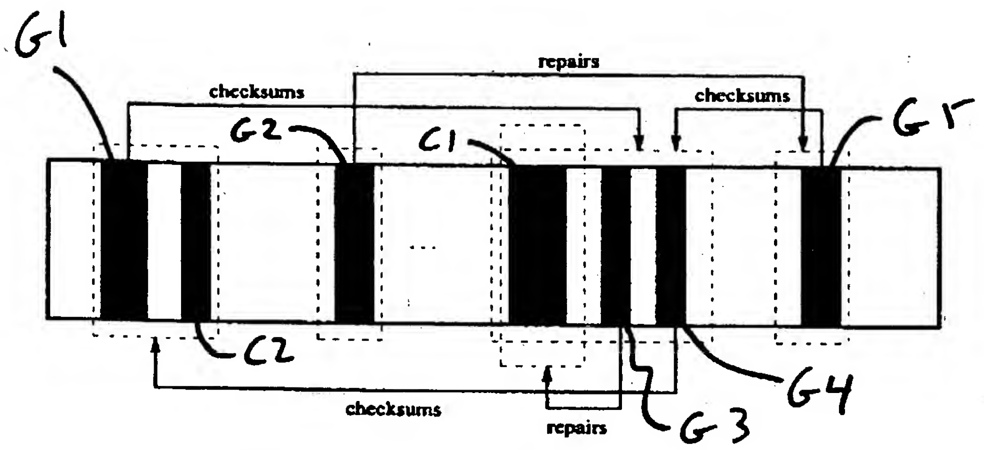
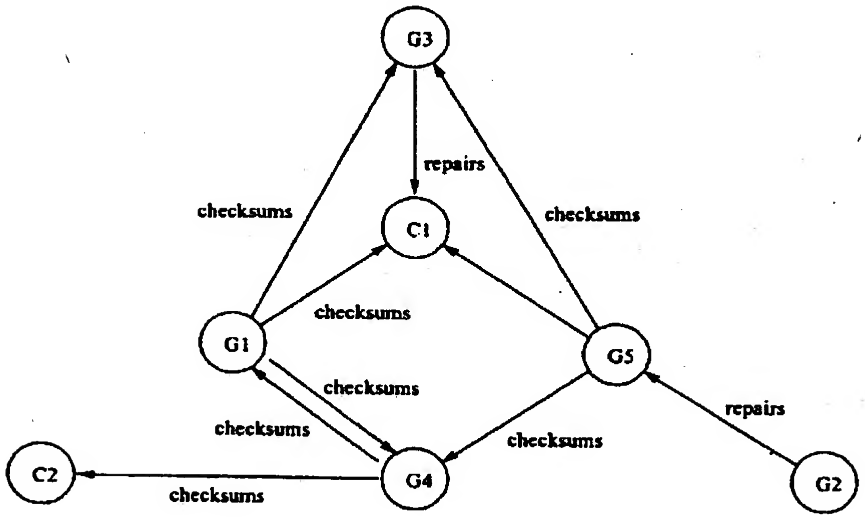


FIG. 9F



(a) Memory layout of the guarded program



(b) The corresponding guard graph

FIG. 9G

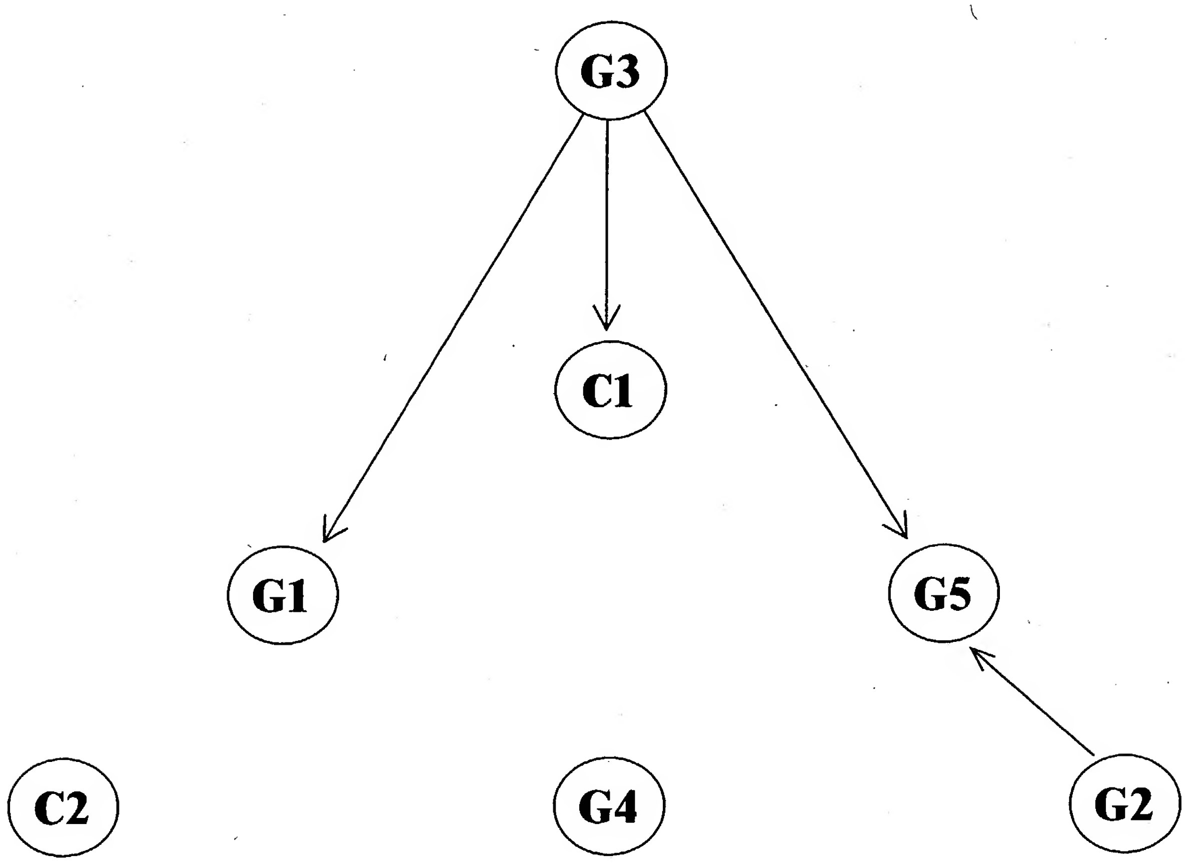


FIG. 9H

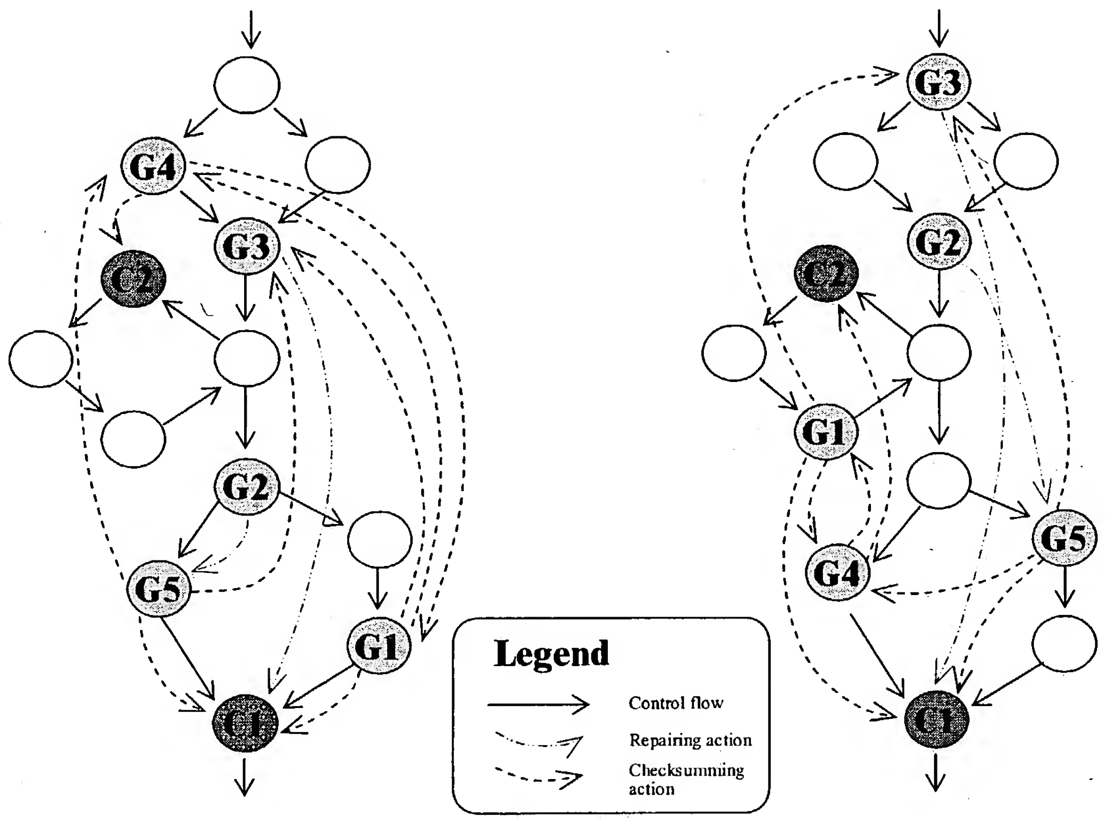

G4
G3
C2
G2
G5
G1
C1
Legend
Control flow
Repairing action
Checksumming action
G3
G2
C2
G1
G5
G4
C1

FIG. 9I

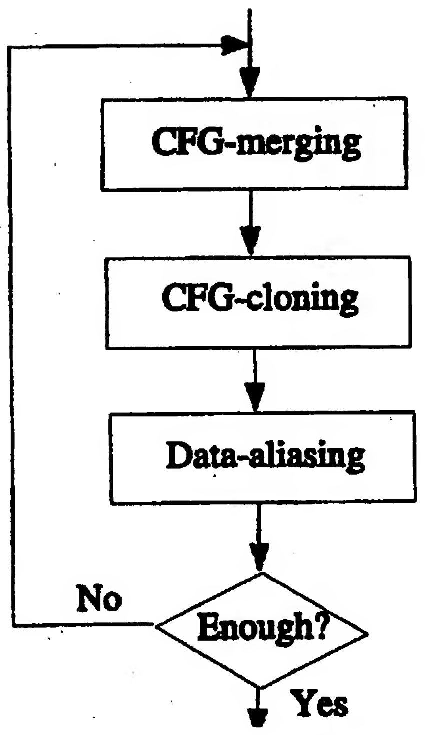


FIG. 10

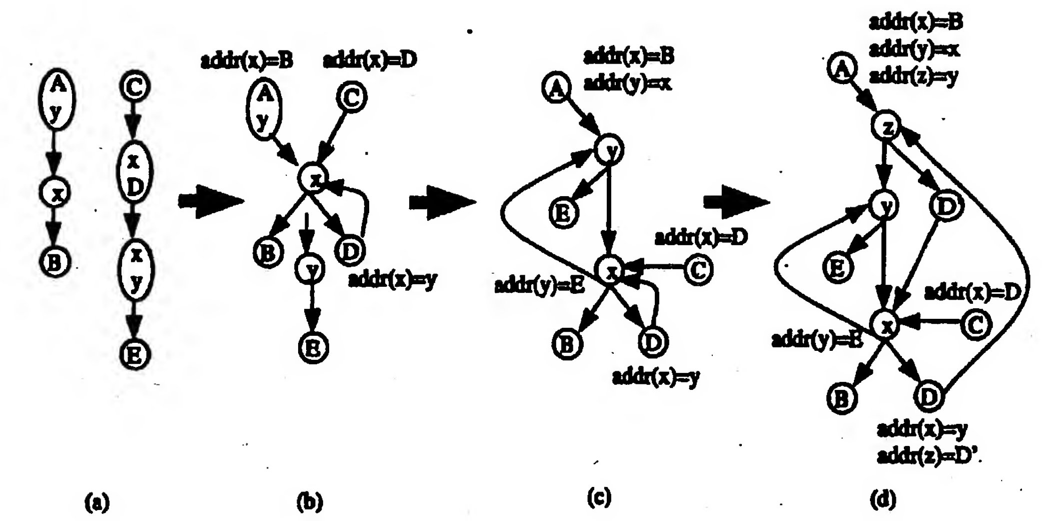
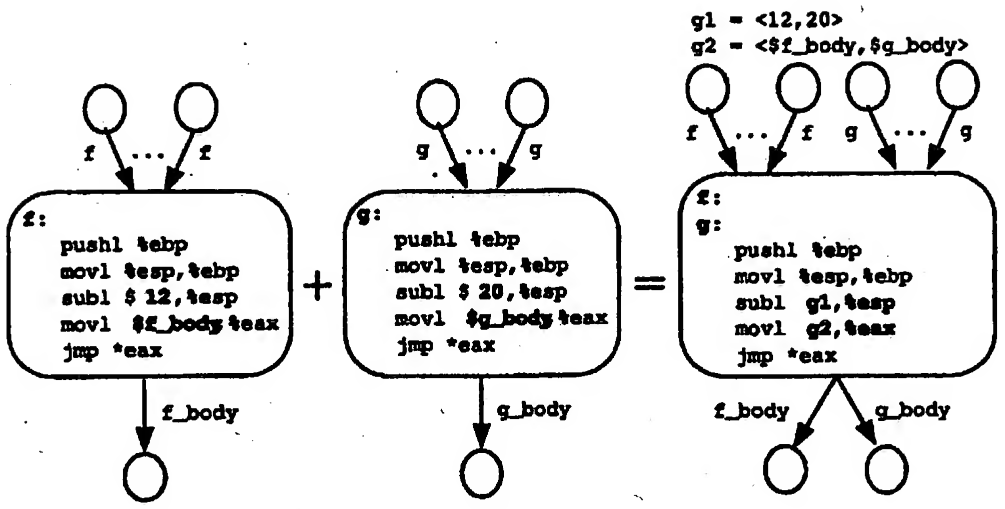

addr(x)=B
addr(x)=D
addr(x)=y
addr(x)=B
addr(y)=x
addr(x)=D
addr(y)=E
addr(x)=y
addr(x)=B
addr(y)=x
addr(z)=y
addr(x)=D
addr(y)=E
addr(x)=y
addr(z)=D'
(a)
(b)
(c)
(d)

FIG. 11



```
f:
    pushl %ebp
    movl %esp,%ebp
    subl $ 12,%esp
    movl  $f_body %eax
    jmp *eax
```

```
g:
    pushl %ebp
    movl %esp,%ebp
    subl $ 20,%esp
    movl  $g_body %eax
    jmp *eax
```

```
f:
g:
    pushl %ebp
    movl %esp,%ebp
    subl  g1,%esp
    movl  g2,%eax
    jmp *eax
```

FIG. 12

```
main:
            movl      $38, g2                      // g2=38                <38>
            movl      $main_2+30, %eax
            subl      g2, %eax
            movl      %eax, g3                     // g3=main_2+30-g2      <main_2-8>
            jmp       main_1
            // g2=<32>, g3=<main_2-8>
main_1:
pr_fact:
            leal      -4(%esp), %esp
            subl      $30, g2                      // g2=g2-30             <8,4>
            movl      %ebp, (%esp)
            movl      g2, %ebp
            addl      %ebp, g3                     // g3=g3+g2             <main_2,pr_fact_1>
            movl      %esp, %ebp
            subl      g2, %esp
            movl      g3, %eax
            jmp       *%eax
            // g1=<,?> g2=<8,4>, g3=<main_2,pr_fact_1>
main_2:
            addl      $26, g2                      // g2=g2+26             <34>
            leal      -8(%ebp), %eax
            leal      -4(%esp), %esp
            movl      %eax, (%esp)
            leal      -4(%esp), %esp
            movl      $str1, (%esp)
            leal      -4(%esp), %esp
            movl      $next1, (%esp)
            jmp       scanf
            // g2=<34>
next1:      // client start
            addl      $8, %esp
            leal      -4(%esp), %esp
            movl      $pr_fact_1-38, %eax
            addl      g2, %eax
            movl      %eax, g3                     // g3=g2+pr_fact_1-38   <pr_fact_1-4>
            movl      -8(%ebp), %eax
            movl      %eax, (%esp)
            leal      -4(%esp), %esp
            movl      $next2, (%esp)
            jmp       pr_fact
            // g2=<34>, g3=<pr_fact_1-4>
end1:       // client end
next2:
            addl      $4, %esp
            movl      %ebp, %esp
            movl      (%esp), %ebp
            leal      4(%esp), %esp
            leal      4(%esp), %esp
            j,mp      *-4(%esp)
pr_fact_1:            // guard installation site
            movl      $100, g1
            movl      $next1, %eax
            jmp       guard1_1
            // g1=<?>
guard1_1:
```

FIG. 13A

```
                cmpl    $end1, %eax
                jg      guard1_2
                jmp     guard1_3
                // g1=<?>
guard1_3:
                movl    g1, %ecx
                addl    (%eax), %ecx
                movl    %ecx, g1
                addl    $3, %eax
                jmp     guard1_1
                // g1=<?>
guard1_2:
                // end of checksumming: (g1, G1)                    // g1=<G1>
                movl    $-G1+1, %eax
                addl    g1, %eax
                movl    %eax, -4(%ebp)
                jmp     .L94
                // g1=<G1>
.L94:
                leal    8+G1(%ebp), %eax
                subl    g1, %eax
                movl    (%eax), %eax
                cmpl    $1, %eax
                jg      .L96
                jmp     .L98
                // g1=<G1>
.L96:
                movl    -4(%ebp), %eax
                imul    8(%ebp), %eax
                movl    %eax, -4(%ebp)
                subl    $1, 8(%ebp)
                jmp     .L94
.L98:
                leal    -4(%esp), %esp
                movl    -4(%ebp), %eax
                movl    %eax, (%esp)
                leal    -4(%esp), %esp
                movl    $str2, (%esp)
                leal    -4(%esp), %esp
                movl    $next3, (%esp)
                jmp     printf
next3:
                addl    $8, %esp
                movl    %ebp, %esp
                movl    (%esp), %ebp
                leal    4(%esp), %esp
                leal    4(%esp), %esp
                jmp     *-4(%esp)
```

FIG. 13B

```
main:
          movl     $38, g2                     // g2=38                  <38>
          movl     $scanf-1000, g5             // g5=scanf-1000          <scanf-1000>
          movl     $main_2+30, %eax
          subl     g1, %eax
          movl     %eax, g3                    // g3=main_2+30-g2        <main_2-8>
          jmp      main_1
          // g2=<38>, g3=<main_2-8>, g5=<scanf-1000>
main_1:
pr_fact:
          leal     -4(%esp), %esp
          subl     $30, g1                     // g2=g2-30               <8,4>
          movl     %ebp, (%esp)
          movl     g2, %ebp
          addl     %ebp, g3                    // g3=g3+g2               <main_2,pr_fact_1>
          movl     %esp, %ebp
          subl     g2, %esp
          movl     $1000, %eax
          addl     g5, %eax
          movl     %eax, g4                    // g4=g5+1000             <scanf>
          movl     g3, %eax
          jmp      *%eax
          // g1=<,?> g2=<8,4>, g3=<main_2,pr_fact_1., g4=<scanf,>
main_2:
          addl     $26, g2                     // g2=g2+26               <34>
          movl     $next1-794320, %eax
          addl     g2, %eax
          movl     %eax, g1                    // g1=g2+next1-794320     <next1-794286>
          leal     -8(%ebp), %eax
          leal     -4(%esp), %esp
          movl     %eax, (%esp)
          leal     -4(%esp), %esp
          movl     $str1, (%esp)
          jmp      main_2_1
          // g1=<next1-794286>, g2=<34>, g4=<scanf>
next1:    // client start
          addl     $8, %esp
          leal     -4(%esp), %esp
          movl     $pr_fact_1-38, %eax
          addl     g2, %eax
          movl     %eax, g3                    // g3=g2+pr_fact_1-38     <pr_fact_1-4>
          addl     $next2-794286-next1, g1     // g1=g1+next2-794286-next1 <next2-794286>
          movl     $pr_fact, g4                // g4=pr_fact             <pr_fact>
          movl     -8(%ebp), %eax
          movl     %eax, (%esp)
          jmp      next1_1
          // g1=<next2-794286>, g2=<34>, g3=<pr_fact_1-4>, g4=<pr_fact>
next1_1:
main_2_1:
.L98_1:
          leal     -4(%esp), %esp
          addl     $794286, g1                 // g1=g1+794286           <next1>
          movl     g1, %eax
          movl     %eax, (%esp)
          movl     g4, %eax
          jmp      *%eax
```

FIG. 14A

```
                // g1=<next2,next1,next3>, g2=<34,34,>, g3=<pr_fact_1-4,,>,
                // g4=<pr_fact,scanf,printf>, g5=<?>
end1:           // client end
next2:
                addl        $4, %esp
                movl        %ebp, %esp
                movl        (%esp), %ebp
                leal        4(%esp), %esp
                leal        4(%esp), %esp
                jmp         *-4(%esp)
pr_fact_1:      // guard installation site
                movl        $100, g1                    //                      <?>
                movl        $next1, %eax
                jmp         guard1_1
                // g1=<?>
guard1_1:
                cmpl        $end1, %eax
                jg          guard1_2
                jmp         guard1_3
                // g1=<?>
guard1_3:
                movl        g1, %ecx
                addl        (%eax), %ecx
                movl        %ecx, g1
                addl        $3, %eax
                jmp         guard1_1
                // g1=<?>
guard1_2:
                // end of checksumming: (g1,G1)
                movl        $printf-G1, %eax
                addl        g1, %eax
                movl        %eax, g4                    // g4=g1+printf-G1
                movl        $-G1+1, %eax
                addl        g1, %eax
                movl        %eax, -4(%ebp)
                jmp         .L94
                // g1=<g1>, g4=<printf>
.L94:
                leal        8+G1(%ebp), %eax
                subl        g1, %eax
                movl        (%eax), %eax
                cmpl        $1, %eax
                jg          .L96
                jmp         .L98
                // g1=<g1>, g4=<printf>
.L96
                movl        -4(%ebp), %eax
                imul        8(%ebp), %eax
                movl        %eax, -4(%ebp)
                subl        $1, 8(%ebp)
                jmp         .L94
                // g4=<printf>
.L98:
                addl        $-G1+next3-794286, g1       // g1=g1-G1+next3-794286    <next3-794286>
                leal        -4(%esp), %esp
                movl        -4(%ebp), %eax
```

FIG. 14B

```
            movl        %eax, (%esp)
            leal        -4(%esp), %esp
            movl        $str2, (%esp)
            jmp         .L98_1
            // g1=<next3-794286>, g4=<printf>
    next3:
            addl        $8, %esp
            movl        %ebp, %esp
            movl        (%esp), %ebp
            leal        4(%esp), %esp
            leal        4(%esp), %esp
            jmp         *-4(%esp)

```

FIG. 14C

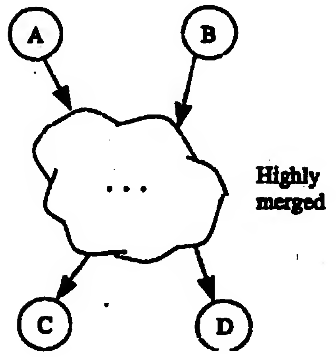


FIG. 15

FIG. 16

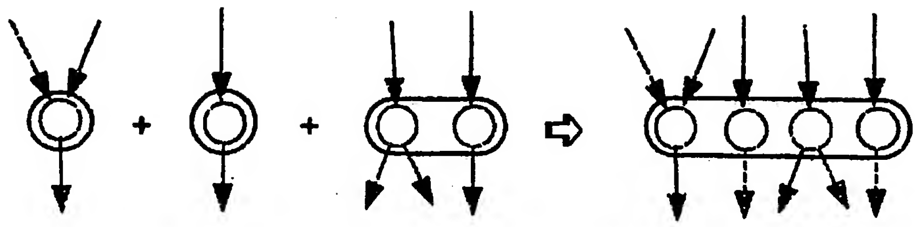


FIG. 17

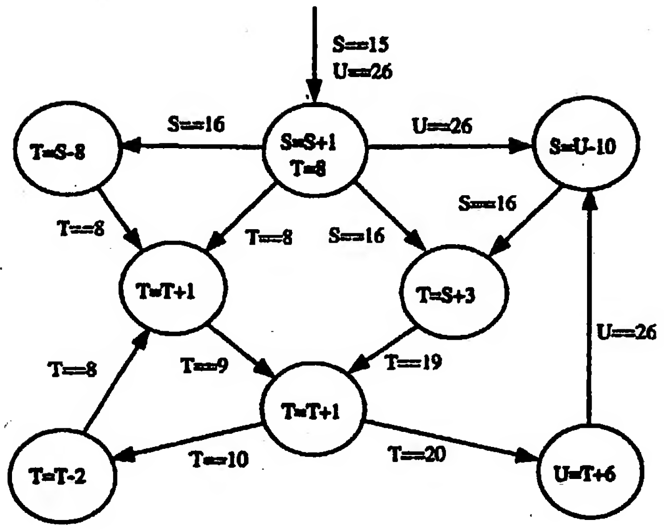

S==15
U==26
S=S+1
T=8
S==16
U==26
T=S-8
S=U-10
T==8
T==8
S==16
S==16
T=T+1
T=S+3
U==26
T==8
T==9
T==19
T=T+1
T=T-2
T==10
T==20
U=T+6

FIG. 18

```
Procedure precompute (variable v, link-nodes {x_1,...,x_n},
                              constants {c_1,...,c_n})

    Set live(v,x_i) = TRUE and value(v,x_i) = c_i for all i = 1,...,n;

    While (there is a link-node x and a variable v such that
            value(v,x) is defined but done(v,x)==FALSE)

        Let X = {x_1,...,x_n} be the entire set of link-nodes in the
                same basic block, say B, that contains x;
        Let t be the point within B for inserting evolve( );

        If (no evolve( ) has previously been chosen for B)
            Choose a mathematical function evolve(U), where U is a (possibly
              empty) set of (new) global variables, with the following properties:
            (1)   No u in U is reserved (i.e. no u in U that, for some z in pred-
                  links(X), live(u,z)==TRUE but value(u,z) is undefined);
            (2)   For any x_i in X where value(v,x_i) is defined, the common
                  evolve(U) is able to fulfill value(v,x_i) by setting value(u,z)
                  appropriately, for any u in U and z in pred-links(x_i) with
                  unseen(u,z,t)==TRUE;
            (3)   For all x_i in X where value(v,x_i) is undefined, the same
                  evolve(U) is possible to fulfill any desired value for any future
                  definition of value(u,x_i);

        End if

        If (evolve( ) is newly chosen and is not the trivial identity function)
            Insert code at t of B for computing v=evolve(U);
        End if

        For (all u in U and z in pred-links(x) where unseen(u,z,t)==TRUE)
            Set live(u,z) = TRUE and value(u,z) equal to some values such that these
            new values, together with other values of U already set along the paths to t
            through z, satisfy value(v,x)==evolve(U);
        End for

        For (each x_i in X where value(v,x_i) is underfined, and each u in U and z in
        pred-links(x_i) where unseen(u,z,t)==TRUE)
            Set live(u,z) = TRUE;
        End for

        Set done(v,x) = TRUE;
        End while
End procedure
```

FIG. 19A

```
Procedure unseen(variable v, link-node z, program point t)
    If (value(v,z) is undefined, and at the basic block that contains t, v is not defined in the
    code before t)
        return TRUE;
    Else
        return FALSE;
    End procedure
```

FIG. 19B

EJ915569813US
P00620US00 (19232.0003)

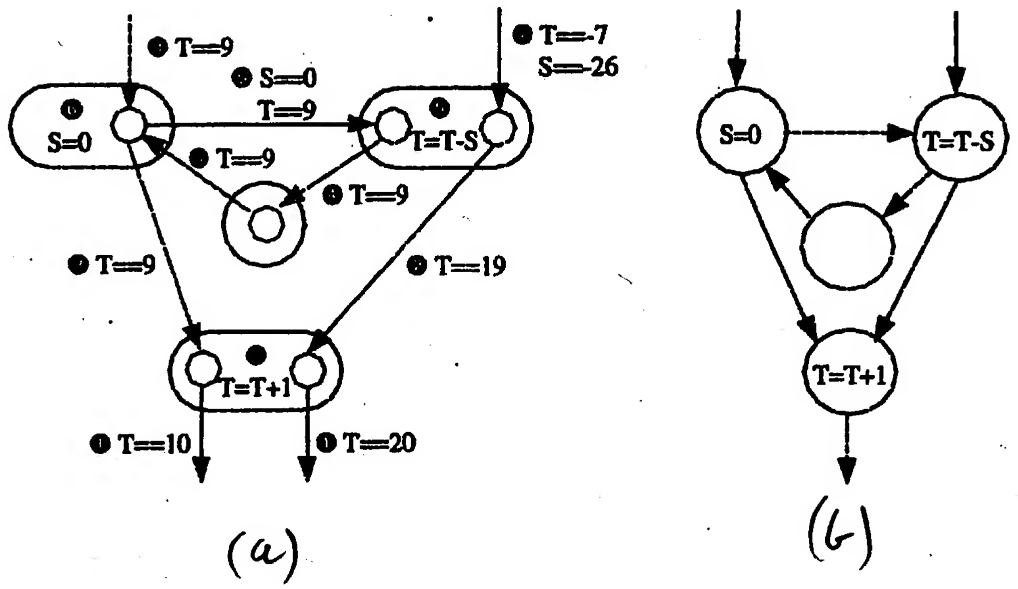


FIG. 20

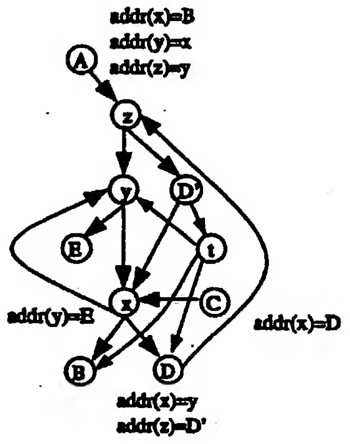

addr(x)=B
addr(y)=x
addr(z)=y
A
z
y
D'
E
t
x
C
addr(y)=E
addr(x)=D
B
D
addr(x)=y
addr(z)=D'

FIG. 21

```
// (a) before change
        . . .
        jmp         .L24

// (b) after change
        . . .
        cmpl        %eax, (%esp)         // args are randomly selected
        jl          .L013                // clone
        jmp         .L24
```

FIG. 22

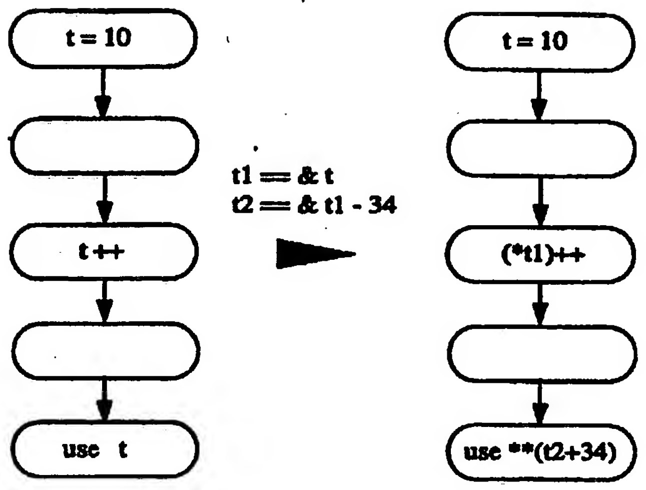


FIG. 23

tag	*sh f*	*E(len + sh f)*	*E(msg + sh f)*

FIG. 24

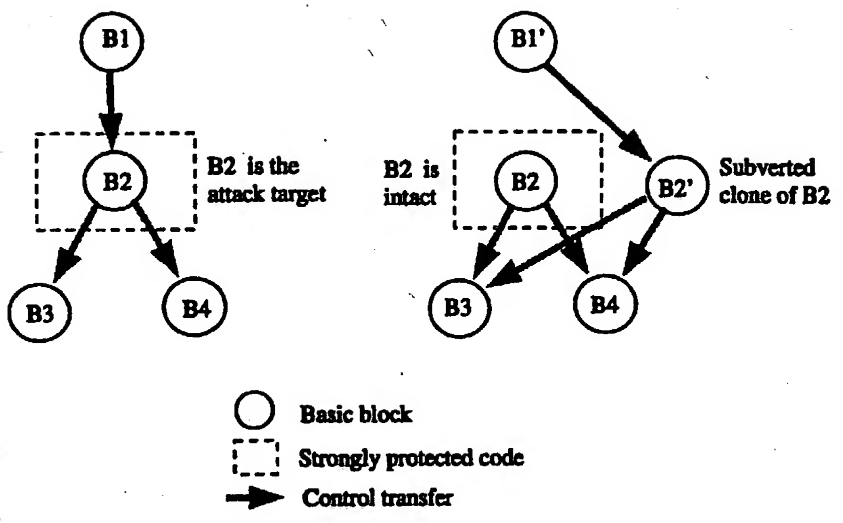

B1
B2
B3
B4
B2 is the attack target
B1'
B2 is intact
B2
B2'
Subverted clone of B2
B3
B4
Basic block
Strongly protected code
Control transfer

FIG. 25

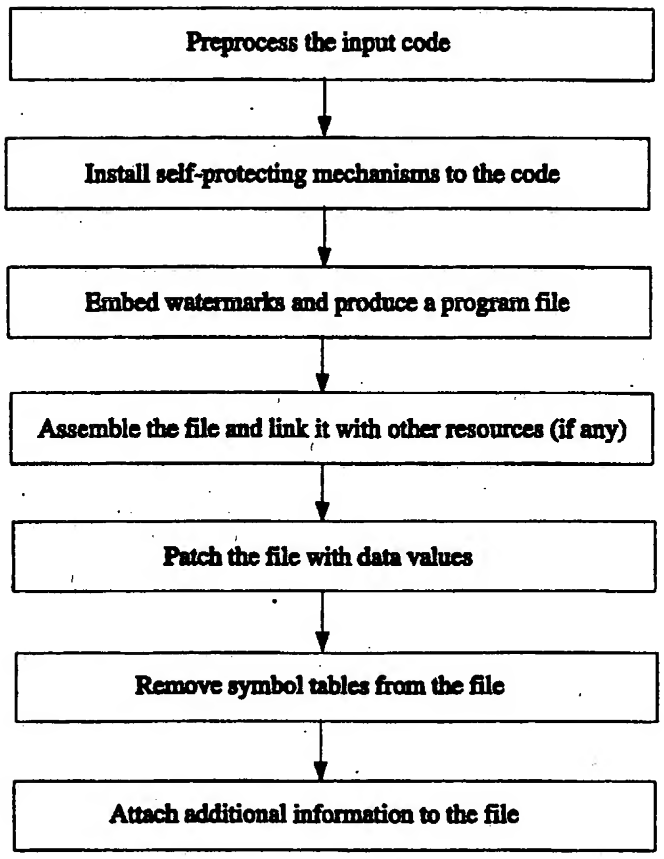

Preprocess the input code
Install self-protecting mechanisms to the code
Embed watermarks and produce a program file
Assemble the file and link it with other resources (if any)
Patch the file with data values
Remove symbol tables from the file
Attach additional information to the file

FIG. 26

High-level instructions	*Simpler instructions*
call *operand* next: . . .	pushl $next jmp *operand* next: . . .
ret	leal 4(%esp), %esp jmp * −4(%esp)
enter	pushl %ebp movl %esp, %ebp
leave	movl %ebp, %esp popl %ebp
pushl *operand*	leal −4(%esp), %esp movl *operand*, (%esp)
pop1 *operand*	movl (%esp), *operand* leal 4(%esp), %esp

FIG. 27

Software Program Code	Encrypted customization parameters	Digital signature

FIG. 28

Wrong password		*After "bypassing" password*
password: **abc** Invalid password!		password: **abc** n ? **100000** next prime = 100001
Right password		*After further "corrections"*
password: **opensesame** n ? **100000** next prime = 100003		password: **abc** n ? **100000** Segmentation Fault (core dumped)

FIG. 29

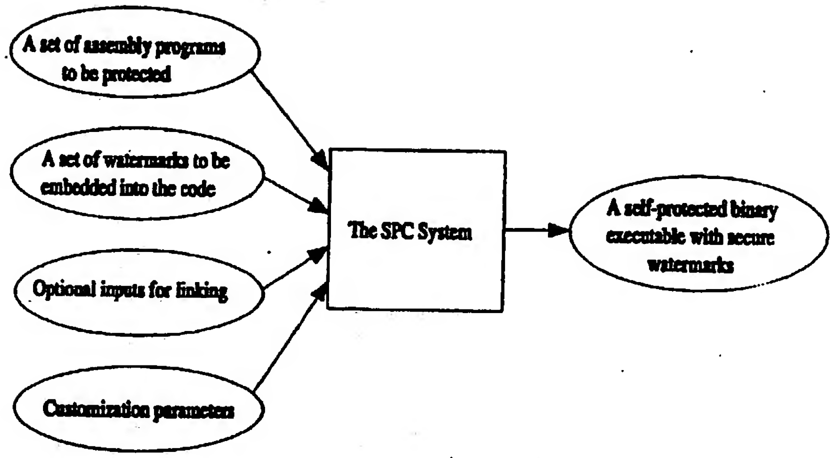

A set of assembly programs to be protected
A set of watermarks to be embedded into the code
Optional inputs for linking
Customization parameters
The SPC System
A self-protected binary executable with secure watermarks

FIG. 30